PGF / TikZ

Willi Egger

Abstract
For those who are looking for an alternative for external graphic drawing tools, PGF / Ti*k*Z offers a wealth of possibilities. PGF is a macro-package that, together with its user interface Ti*k*Z, comprises a kind of "graphics language" to build graphics inside the text as inline graphics or as pictures of larger size. PGF is a macro-package originally written for LaTeX. In the meantime it is also available for use within ConTeXt. The package comes with a large set of libraries for different kinds of graphics. There is extensive documentation and a tutorial. For support a mailing list and web-site are available. Users of the package with ConTeXt have to install the xkeyval package version 1.8. PGF and Ti*k*Z are distributed under the GNU Public License version 2.

Introduction

The drawing environment is called PGF. PGF is the acronym for "Portable Graphics Format". The user interface is called Ti*k*Z. Ti*k*Z stands for "Ti*k*Z ist *kein* Zeichenprogramm" meaning it is not an interactive drawing/painting program. The package provides a kind of "graphics language" with which to program a graphic comparable to programming the text in TeX.

PGF / Ti*k*Z is a macropackage that was originally written for LaTeX. In the meantime ConTeXt users can also profit from this package and use it within ConTeXt.

The package is written and maintained by Till Tantau, professor at the Institut für Theoretische Informatik, Universität Lübeck.

The macropackage is distributed under the GNU Public License version 2.

Installation

LaTeX
Because the package is basically a LaTeX-package, it is probably already installed on your system. Otherwise install it as usual in the TeX tree e.g. from CTAN.

ConTeXt
The easiest way to install the package under ConTeXt is to download it and unpack it in a temporary folder. Move the contents of the *generic* folder to `...\tex\generic`. Move the contents of the folder *context* to `...\tex\context\third` (the third party module folder). In order to get the package working you need to install the xkeyval version 1.8 package written by Hendri Adriaens. Download it from CTAN. Install the files either in `...\tex\generic` or `...\tex\latex`. Move the folder *doc* to `.../doc`. Run mktexlsr in order to update the file-database.

Plain TeX
Provided that you use a full installation of TeX, like the TeX-live distribution, the package might be installed already. Otherwise, install the necessary files contained in the archive into the respective folders of the TeX-tree. For use of the package under Plain TeX you need also the xkeyval package version 1.8 or above by Hendri Adriaens and the xcolor package version 2.0 by Uwe Kern.

Syntax differences

The graphics language syntax for the different TEX-environments is generally set up in such a way that the user can use the commands as he is accustomed to do in his TEX-environment:

In order to use the package, it must first be loaded:

LaTEX	ConTEXt	Plain TEX
`\usepackage{tikz}`	`\usemodule[tikz]`	`\input tikz.tex`

Starting a sequence of commands for a graphic:

LaTEX	ConTEXt	Plain TEX
`\begin{tikzpicture}`	`\starttikzpicture`	`\tikzpicture`
`...`	`...`	`...`
`\end{tikzpicture}`	`\stoptikzpicture`	`\endtikzpicture`

Structure of the package

The package is built in three layers: system, basic and front end. The system-layer provides the highest abstraction level, i.e. it provides the translations of commands into `\special` -commands as required by the different driver environments (dvips, dvipdfm, pdftex). This layer is not intended for use by the user.

The second layer is called basic-layer. This layer provides a set of commands for building complex graphics without being obliged to use the syntax of the system layer.

Lastly, the front end layer provides a set of commands which make the use of the basic layer more convenient. The most natural front end is Ti*k*Z. For illustration, if you wanted to draw e.g. a triangle with the basic layer you would have to issue up to 5 commands, whereas with the Ti*k*Z-front end it will suffice to say `\draw (0,0)--(1,0)--(1,1)--cycle;`.

Ti*k*Z

Defining points in a graphic

There are different possibilities to define a point in a graphic. First, a point can be defined by giving the coordinates in dimensions known to TEX inside a pair of round brackets e.g. (2cm,10pt). This represents the xy-coordinate system. Giving three dimensions inside a pair of round brackets (1,1,1) will invoke the xyz-coordinate system. Omitting the unit of a dimension will cause Ti*k*Z to use the predefined dimension of 1cm. A third way of defining a point is to indicate a vector in the form of (30:1cm). This will cause Ti*k*Z to move 1cm in the direction of 30 degrees. Finally, points can be specified relative to another point. e.g. (1,0) ++(1,0) ++(0,1) specifies the following movements: (1,0) (2,0) (2,1). One can also define a relative point by adding a single +: (1,0) +(1,0) +(0,1) - which defines the following coordinates: (1,0) (2,0) (1,1). The difference is, that the points defined with a single + will not change the current point which is in the example (1,0).

Paths

Paths consisting of a series of straight and curved elements are defined similar to MetaPost. Path elements do not necessarily need to be connected to each other. Actions on (closed) paths are draw, fill, shade and clip. These actions can be applied to paths in any combination.

Grouping

Within a Ti*k*Z-picture grouping of elements can be achieved by using scopes

LaTEX: `\begin{scope}[`*options*`]...\end{scope}` and

ConTEXt: `\startscope[`*options*`]...\stopscope`.

The application of styles enables the user to apply options defined outside the graphic to (part of) the graphic.

Transformations
Shifting an object can be done with `\shift{1,2}` or `\shift{+1,2}` i.e. with relative positioning. There is also `xshift=10pt` and `yshift=1cm`. Objects can be rotated with the command `rotate` or rotated around a given point with `rotated around`. `scale=2` will scale the object or picture by the given factor. There is also `xscale` and `yscale` for scaling in a single direction.

For-loops
LaTeX-users can issue the built in loop constructs or the `\multido` command from pstricks. ConTeXt-users can use the `\dorecurse{}{}` looping mechanism. Ti*k*Z offers yet another for-loop, based on series represented by dimensionless real numbers, which are given as `{1,2,3}` or `{1,...,10}` or even `{1,3,...,20}`. In case two figures are given before the ellipsis, the difference between the two figures is used as step. The basic command is

```
\foreach \variablename in {...series...}
    \draw(\variablename pt, -1pt) -- (\variablename pt,1pt)
```

Text and Labels
Text can be added to any given point of a path. When Ti*k*Z encounters the keyword `node` during the construction of a path, the elements of the node will be added as a TeX-box after the complete path is ready. A node consists of the keyword `node` followed by options between square-brackets and followed by the text enclosed in curly braces.

Node-options are numerous: `left/right/below/above =`*dimension*, anchors are `north, south, west, east, north east` ... Furthermore text may be positioned along the path with the option `slope`. Texts may be moved towards the beginning or end of a path by `very near` or `near end`. These options can be combined with `above` etc. and `sloped`.

Libraries

Ti*k*Z comes with a large set of specialized libraries. Each library needs to be loaded separately: Libraries are loaded by `\usetikzlibrary{library-name}` in LaTeX and `\usetikzlibrary[library-name]` in ConTeXt.

— Arrow Tip Library, *arrows*
  This library contains a large pallet of different tip-styles like triangular tips, barbed and bracket-like, circle and diamond like, partial tips and line caps.
— Automata Drawing Library, *automata*
  For the drawing of finite automata and Turing machines.
— Background Library, *backgrounds*
  Various background types are provided.
— Entity-Relationship Drawing Library, *er*
  This library will be loaded for drawing E/R diagrams.
— Mind-map Library, *mindmap*
  For those who need a nice presentation of their mind-maps, this library is useful.
— Pattern Library, *patterns*
  For filling shapes with pattern this library is loaded.
— Petri-Net Library, *petri*
  For the construction of Petri-nets this library is needed.
— Plot Handler Library, *plothandlers*
  The plot handler library is loaded automatically by Ti*k*Z.

– Plot Marks Library, *plotmarks*
  This library defines additional plot marks next to the standard marks, which are *, x and +.
– Shape Library, *shapes*
  The shapes library contains a number of predefined shapes.
– Snake Library, *snakes*
  This library defines a series of non-straight lines like coils, braces, bumps, expanding waves, saw or yes, snake lines.
– To Path Library, *topaths*
  This library is loaded automatically by Ti*k*Z and provides predefined to paths which are used in the to path operation.
– Tree Library, *trees*
  This library provides different styles to draw trees.

Examples

Below are some examples that demonstrate what Ti*k*Z can do. The examples are taken from the manual, and are formatted for ConTEXt. LaTEX-users can best refer to the manual, because examples are given there in LaTEX-code.

Using Ti*k*Z code inline in the text
For drawing a sloped line within the text row you would type

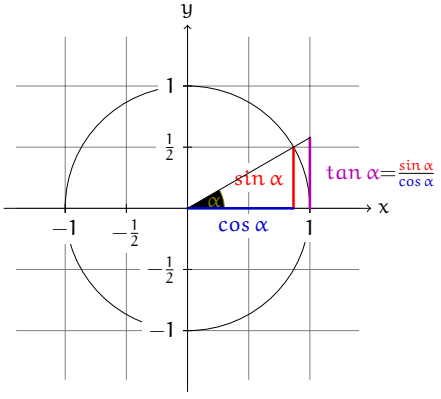
```
\tikz \draw(0pt,0pt) -- (20pt,6pt);
```

and get . Here a big grey dot should be placed . This is coded as

```
\tikz \fill[black!60] circle (1ex);
```

.

Trigonometry



The angle α is 30° in the example ($\pi/6$ in radians). The sine of α, which is the height of the red line, is [$\sin\alpha = 1/2$.] By the Theorem of Pythagoras ...

Figure 1. Trigonometry

```
\usemodule[tikz]
\starttikzpicture[scale=2,cap=round]
    % Local definitions
    \def\costhirty{0.8660256}
    % Colors
    \definecolor[anglecolor][r=.5,g=.5,b=0]
    \definecolor[sincolor][r=1,g=0,b=0]
    \definecolor[tancolor][r=.7,g=0,b=.7]
    \definecolor[coscolor][r=0,g=0,b=.8]
    \definecolor[fillcolor][.625black]
    % Styles
```

```
    \tikzstyle{axes}=[]
    \tikzstyle{important line}=[very thick]
    \tikzstyle{information text}=[rounded corners,inner sep=1ex]
    % The graphic
    \draw[style=help lines,step=0.5cm] (-1.4,-1.4) grid (1.4,1.4);
    \draw (0,0) circle (1cm);
    \startscope[style=axes]
       \draw[->] (-1.5,0) -- (1.5,0) node[right] {$x$} coordinate(x axis);
       \draw[->] (0,-1.5) -- (0,1.5) node[above] {$y$} coordinate(y axis);
       \foreach \x/\xtext in {-1, -.5/-\frac{1}{2}, 1}
       \draw[xshift=\x cm] (0pt,1pt) -- (0pt,-1pt) node[below,fill=white]
           {$\xtext$};
       \foreach \y/\ytext in {-1, -.5/-\frac{1}{2}, .5/\frac{1}{2}, 1}
       \draw[yshift=\y cm] (1pt,0pt) -- (-1pt,0pt) node[left,fill=white]
           {$\ytext$};
    \stopscope
    \filldraw[fill=green!20,draw=anglecolor] (0,0) -- (3mm,0pt) arc(0:30:3mm);
    \draw (15:2mm) node[anglecolor] {$\alpha$};
    \draw[style=important line,sincolor] (30:1cm) -- node[left=1pt,fill=white]
        {$\sin \alpha$} (30:1cm |- x axis);
    \draw[style=important line,coscolor]
        (30:1cm |- x axis) -- node[below=2pt,fill=white] {$\cos \alpha$} (0,0);
    \draw[style=important line,tancolor] (1,0) -- node[right=1pt,fill=white]
        {$ \tan \alpha {\color[black]=}
           \frac{\color[sincolor]\sin \alpha}{\color[coscolor]\cos \alpha}$}
        (intersection of 0,0--30:1cm and 1,0--1,1) coordinate (t);
    \draw (0,0) -- (t);
    \draw[xshift=2.5cm]
    node[right,text width=6cm,style=information text]
    {
    The {\color[anglecolor] angle $\alpha$} is $30^\circ$ in the
    example ($\pi/6$ in radians). The {\color[sincolor]sine of
    $\alpha$}, which is the height of the red line, is
    [{\color[sincolor] \sin \alpha} = 1/2.]

    By the Theorem of Pythagoras ...
    };
\stoptikzpicture
```

Simple organigramma / tree

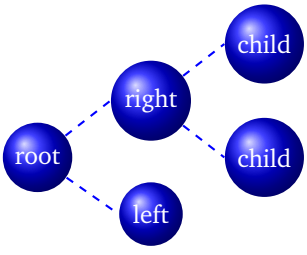


Figure 2. A tree

```
\usemodule[tikz]
\usetikzlibrary[arrows,snakes,backgrounds,trees]
\starttikzpicture[parentanchor=east,child anchor=west,grow=east]
     \tikzstyle{every node}=[ball color=blue,circle,text=white]
     \tikzstyle{edge from parent}=[draw,dashed,thick,blue]
```

```
      \node {root}
      child {node {left}}
      child {node {right}
      child {node {child}}
      child {node {child}}
      };
\stoptikzpicture
```

TikZ in cooperation with GNUplot

One can use GNUplot to calculate the points needed in a path. TikZ will, after reading its instructions, prepare a command file containing GNUplot commands describing the function to be drawn. In a second run GNUplot is envoked with these commands from which a data file is produced. TikZ imports this data file and draws the graph.

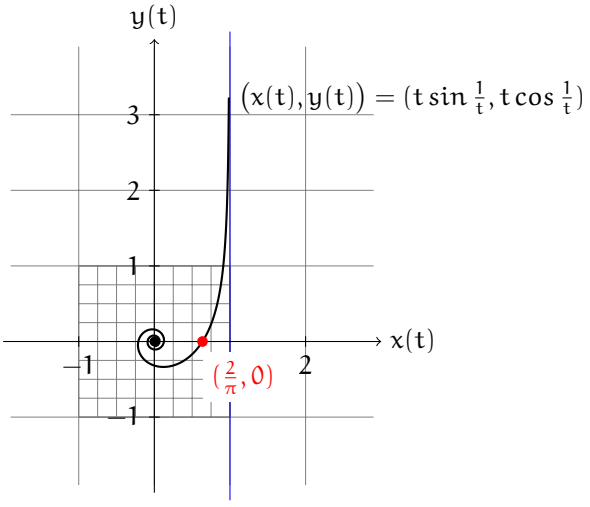


Figure 3. A function calculated by GNUplot

```
\usemodule[tikz]
\starttikzpicture
    \draw[gray,very thin] (-1.9,-1.9) grid (2.9,3.9)
                    [step=0.25cm] (-1,-1) grid (1,1);
    \draw[blue] (1,-2.1) -- (1,4.1);
    \draw[->] (-2,0) -- (3,0) node[right] {$x(t)$};
    \draw[->] (0,-2) -- (0,4) node[above] {$y(t)$};
    \foreach \pos in {-1,2}
         \draw[shift={(\pos,0)}] (0pt,2pt) -- (0pt,-2pt) node[below] {$\pos$};
    \foreach \pos in {-1,1,2,3}
         \draw[shift={(0,\pos)}] (2pt,0pt) -- (-2pt,0pt) node[left] {$\pos$};
    \fill (0,0) circle (0.064cm);
    \draw[thick,parametric,domain=0.4:1.5,samples=200]
    % The plot is parameterized such that there are more samples
    % near the centre.
    plot[id=asymptotic-example]
         function{(t*t*t)*sin(1/(t*t*t)), (t*t*t)*cos(1/(t*t*t))}
         node[right]
         {$\bigl(x(t),y(t)\bigr) = (t\sin \frac{1}{t},t\cos\frac{1}{t})$};
    \fill[red] (0.63662,0) circle (2pt)
```

```
        node [below right,fill=white,yshift=-4pt] {$(\frac{2}{\pi},0)$};
\stoptikzpicture
```

The contents of the file created by GNUplot looks as follows:

```
#Curve 0, 200 points
#x y type
0.00530 -0.06378        i
0.04363 -0.05043        i
0.06711 -0.01790        i
0.06896 0.02170         i
0.05014 0.05606         i
0.01712 0.07631         i
-0.02110 0.07849        i
-0.05579 0.06337        i
-0.08032 0.03512        i
-0.09097 -0.00029       i
-0.08696 -0.03664       i
-0.06987 -0.06850       i
-0.04284 -0.09192       i
-0.00982 -0.10460       i
0.02515 -0.10585        i
0.05841 -0.09629        i
...
0.98511 3.18914 i
0.98543 3.22793 i
```

Internet

More information on PGF and Ti*k*Z can easily be found on the Internet. For a stable release of the package one can visit CTAN or from `http://sourceforge.net/projects/pgf/`.

If you want to get the most recent developments you can fetch the latest version from CVS. The command would be something like

```
cvs -z3 -d:pserver:anonymous@pgf.cvs.sourceforge.net:/cvsroot/pgf
                                                        -co checkout pgf
```

For extensive examples there is a web-site by Kjell Magne Fauske, Norway:

`http://www.fauskes.net/pgftikzexamples`

For support one can join the following mailing-list. Visit

`https://lists.sourceforge.net/lists/listinfo/pgf-users`

for subscription.

Summary

Ti*k*Z is a tool for making various kinds of drawings. For TEX-users the style of setting up such drawings is familiar, because you program a drawing similarly to how you program a TEX-text. Ti*k*Z is the natural front end to lower level PGF functionality. On top of this, it is possible to draw graphs using points generated by GNUplot.

Acknowledgement

I would like to thank Michael Guravage for looking through the text and turn it into correct English.

Willi Egger